Case-based Policy Inference for Transfer in Reinforcement Learning

Ruben Glatt, Felipe Leno da Silva, and Anna Helena Reali Costa

Escola Politécnica da Universidade de São Paulo, Brazil
{ruben.glatt, f.leno, anna.reali}@usp.br

Abstract. This paper introduces a formulation for applying the *Case-Based Reasoning* methodology to knowledge transfer with *Reinforcement Learning* agents. Based on this framework, we propose the *Case-based Policy Inference (CBPI)* algorithm to accelerate learning by selecting similar cases from a library of previously learned tasks with their respective solutions to solve a new target task. *CBPI* guides the training by dynamically selecting and blending policies according to their usefulness for the current target task, progressively switching the control to the new policy as it converges to the optimal behavior. In our experiments, we show the benefits of our algorithm with regards to sample efficiency and performance when compared to vanilla *Q-Learning* and *Probabilistic Policy Reuse* in a grid world domain. Apart from converging faster, *CBPI* can also effectively learn core policies of a given domain and shows to be robust against negative transfer.

Keywords: Reinforcement Learning · Case-based Reasoning · Transfer Learning · Policy Reuse

1 Introduction

A great part of *Artificial Intelligence (AI)* research is concerned with learning how to solve a given task in a most efficient way. In the past, most research efforts have been focused on learning individual tasks from scratch. One way, such tasks can be modeled is as a *Markov Decision Process (MDP)* [18], which is a discrete time stochastic control process. At each time step, an *MDP* is in a state and an agent can perform actions that lead to an update of the state combined with a reward signal.

Many very powerful methods for solving this kind of decision-making processes are specified in the field of *Reinforcement Learning (RL)* [24]. In *RL*, an agent explores the space of possible strategies to solve a task in a given environment, receives a feedback (reward) on the outcome of the actions it takes and deduces a behavior policy from its observations over time. The goal of the agent is to determine a policy π that maps each state s to an action a, which maximizes the accumulated reward R over a given horizon. But, although *RL* has been successfully used to autonomously learn how to solve complex tasks,

like classic board games [26] or robot soccer [23], learning to solve a new task with good results still takes a relative long time. This is due to the fact that agents applying *RL* techniques require a large number of samples of interactions with the environment to infer an effective solution policy even for simple tasks. However, with the increasing success in the area (e.g. autonomous helicopter flight [16] or Atari computer games [15]), the focus has shifted to more complex agents that can learn to solve not only one, but multiple tasks with the help of using a priori gathered knowledge in order to progressively reduce the sample complexity.

The field of *Transfer Learning (TL)* emerged from the need to solve these kind of problems and in recent years many efforts have been directed on expanding *Machine Learning (ML)* algorithms with the ability to transfer knowledge, experience or skills to allow learning of multiple tasks more efficiently [17]. The idea of using accumulated knowledge in this way is inspired by the human transfer learning abilities which work quite similar. The goal is to create intelligent agents that can learn to master varied tasks in a single or various domains by engaging in continuous or even lifelong learning [27]. An overview of approaches that reuse knowledge from previously learned tasks utilizing *RL* approaches is given by Taylor & Stone [25], but the interest is still growing and new interesting concepts are emerging frequently, as for example transferring knowledge in the form of options [12], by transferring artificial neural network weights [8] or through advice from other agents [21].

Another important concept to benefit from previously acquired knowledge is described in the *Case-based Reasoning (CBR)* approach [1], which describes a methodology to build computational models to reuse existing knowledge in a general manner [30]. The guiding assumption for *CBR* is that similar problems have similar solutions. *CBR* has been shown to be very successful and has been widely applied in a number of fields [5]. A high-level description of the *CBR* cycle describes four stages during the learning of a new task with previously acquired knowledge [1, 11]. First, the agent has to *RETRIEVE* the most similar cases from its knowledge base and then *REUSE* the contained information to propose a solution to the new task. After that, the proposed solution is *REVISED* to find a final solution to the target task. In the end, the agent can decide if it should *RETAIN* the gathered solution in the knowledge base.

Previous works in *RL* have applied similar methodologies as *CBR*, often without making a direct connection. One example of this is the instance-based learning approach from McCallum [14], where a memory of exploration traces is stored to overcome the state aliasing issues. Another one is Kuhlmann and Stone [13], who introduce a graph-based method for identifying previously encountered games to automate domain mapping for value function transfer.

In this paper, we are applying the *CBR* methodology to a decision making problem that can be solved through *RL* methods. We describe a general framework and introduce a concrete implementation termed *Case-based Policy Inference (CBPI)* to learn solutions for a given targt task. The training in our framework is guided by the Π-Inference algorithm that we propose here. The

validity of the approach is shown in an experimental evaluation, where we compare our approach with regular *Q-Learning* [29] and *Probabilistic Policy Reuse (PPR)* [6] in a simple robot navigation domain. We show the general superiority over those algorithms specifically with regard to increased performance and efficiency of *CBPI* under certain conditions.

The remainder of this paper is organized as follows: Section 2 describes our interpretation of a general *CBR* framework. In Section 3, we propose the *CBPI* approach, followed by Section 4, where we report the conducted experiments and analyse the outcome. Section 5 then compares our approach to related works, and, finally, Section 6 concludes the paper and points towards further directions.

2 Case-Based Reasoning in Reinforcement Learning terminology

In this section we describe the *CBR* framework in *RL* terminology. As is common for sub-fields of a research area, *CBR* has evolved its own terminology and perspectives, making it less tangible for other communities. Therefore, we attempt to provide a formulation that defines the *CBR* framework using *RL* terminology with the goal of making it more accessible for the *RL* community and making *RL* more attractive to the *CBR* community.

In *CBR* terminology, a case describes a problem and its solution. A case base is comprised of a number of retained cases, which have been learned in the past and whose solutions can be reused to solve future problems.

In *RL*, a *case* contains the task (problem), defined by an *MDP*, and a policy associated with that task (solution) and can be described as

$$\vartheta := \langle \Omega_\vartheta, \pi_{\Omega_\vartheta} \rangle, \tag{1}$$

where ϑ stands for the *case*, Ω_ϑ represents the task, and π_{Ω_ϑ} represents a policy, that should ideally be an optimal solution for Ω_ϑ.

A *case base* can be defined as a *library* in *RL* and formally described as

$$\mathcal{L} := \{\vartheta_0, \vartheta_1, ..., \vartheta_n\}, \tag{2}$$

where the ϑ_i stand for individual cases that have successfully been solved and added to the library. Each of those cases is ideally distinct from each other to avoid duplicate information.

The cyclic process in *CBR* integrates solving a new task and learning from experience while building a case base over time. Here, we are providing a general view on applying the *CBR* cycle to *RL*. A visualization of this process is sketched out in Figure 1. We are leaving the definitions intentionally wide so that adaptions to other variants are possible, as for example by Aha [2], who introduces an additional *REVIEW* step between *REVISE* and *RETAIN*, or by Hüllermeier [9] who also describes a framework but focuses on the *RETAIN* stage and proposes to build a *credible* set that contains some (possibly all) solutions for a given problem.

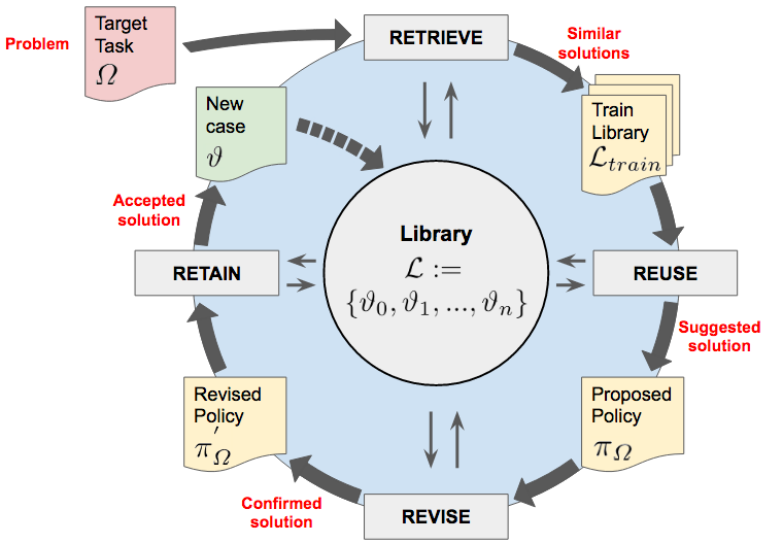

Fig. 1. A view on the *CBR* cycle adapted to RL terminology.

RETRIEVE. For every new target task Ω that an agent wants to learn, it analyses the existing knowledge in the library $\mathcal{L}$ because it is not necessarily beneficial to use the whole library $\mathcal{L}$ when learning a new task. Therefore, only the cases with the most similar tasks should be selected from $\mathcal{L}$ according to a similarity function

$$sim_{task} : \Omega \times \Omega \rightarrow [0, 1] \tag{3}$$

to build a more confined subset $\mathcal{L}_{train} \subseteq \mathcal{L}$.

The function that builds the training library can be described as

$$\mathcal{L}_{train} \leftarrow RETRIEVE(\mathcal{L}, \Omega), \tag{4}$$

and formalized as

$$\mathcal{L}_{train} \leftarrow \{\vartheta | \vartheta \in \mathcal{L} \text{ and } sim_{task}(\Omega, \Omega_\vartheta) \geq \sigma_{task}\}, \tag{5}$$

where σ_{task} defines a threshold factor for controlling the size of the library.

REUSE. In this stage, the training library is used to support the exploration of possible solutions to suggest a solution that helps to solve the task. The training here is guided by a strategy that is built on the policies of the retrieved similar cases in the training library $\mathcal{L}_{train}$.

It might be necessary to adapt a form of mapping for each case if the state- and/or action-space is different than in the target task. The resulting policy is found by continuously iterating between using the already learned policies and

exploring the current target policy. The function that returns this policy and defines the used strategy can be described as

$$\pi_{\Omega} \leftarrow REUSE(\mathcal{L}_{train}, \Omega). \tag{6}$$

The proposed solution for the target task Ω, a policy π_{Ω}, then defines the behaviour of the agent.

REVISE. Although some of the cases in the library may contain very similar tasks, none of them are expected to be exactly equal to the new target task. For this reason, selecting one (or multiple) policies to guide the training is a good way to get to reasonable solutions fast, but needs to be followed by further training to verify, and if possible improve, the suggested solution. At this stage, the policy is refined using either a new strategy or by following the same strategy as in the *REUSE* stage adapted to a single solution (policy). The function that returns the revised solution $\pi_{\Omega}^{'}$ which in the best case is optimal, is described as

$$\pi_{\Omega}^{'} \leftarrow REVISE(\Omega, \pi_{\Omega}). \tag{7}$$

In *RL*, often the *REUSE* and *REVISE* steps might be integrated in the same function, gradually switching the guiding policy from past experiences to the new policy.

RETAIN. When the training for the new task ends (whether by finding the optimal policy or by meeting a termination condition), it must be decided if the task Ω with the solution $\pi_{\Omega}^{'}$ should be added as a new case to $\mathcal{L}$. This is achieved by comparing $\pi_{\Omega}^{'}$ with the existing solutions from the other cases in the library. The comparison is performed using a policy similarity function

$$sim_{solution} : \Pi \times \Pi \rightarrow [0, 1]. \tag{8}$$

The new case ϑ enters the library $\mathcal{L}$ only if it returns a value smaller than a threshold factor $\sigma_{solution}$ for all policies. This can be described as

$$\mathcal{L} \leftarrow RETAIN(\mathcal{L}, \langle \Omega, \pi_{\Omega}^{'} \rangle), \tag{9}$$

and formalized as

$$\mathcal{L} \leftarrow \begin{cases} \mathcal{L} \cup \langle \Omega, \pi_{\Omega}^{'} \rangle & \text{if } \forall \vartheta \in \mathcal{L} : \ sim_{solution}(\pi_{\Omega}^{'}, \pi_{\Omega_{\vartheta}}) \leq \sigma_{solution} \\ \mathcal{L} & \text{otherwise} \end{cases}. \tag{10}$$

Algorithm. The whole *CBR* cycle for RL can then be described using the definitions above as shown in Algorithm 1.

Algorithm 1 CBR cycle for RL

function CBRRL($\mathcal{L}$, Ω)
    $\mathcal{L}_{train} \leftarrow RETRIEVE(\mathcal{L}, \Omega)$
    $\pi_\Omega \leftarrow REUSE(\mathcal{L}_{train}, \Omega)$
    $\pi_\Omega^{'} \leftarrow REVISE(\Omega, \pi_\Omega)$
    $\mathcal{L} \leftarrow RETAIN(\mathcal{L}, \langle \Omega, \pi_\Omega^{'} \rangle)$

3 Case-based Policy Inference (CBPI)

We propose an implementation of the framework described in Section 2 and coin it *Case-based Policy Inference (CBPI)*. As we are using *Q-Learning* to find a solution, the policy for the new task is based on a Q-table and defined as

$$\forall s \in S : \pi_\Omega(s) = \arg\max_{a \in A} Q_\Omega(s, a). \tag{11}$$

CBPI is based on the idea of creating a generalization of past policies during training and focuses on the *REUSE* and *REVISE* stages of the *CBR* cycle and is described in Algorithm 2. It should not be confused with the term *Case-based Inference (CBI)*, which aims at predicting the right solution for a new target task including assertions about the confidence of those predictions [9].

Algorithm 2 Case-based Policy Inference

function CBPI($\mathcal{L}$, Ω, τ_{policy}, $\Delta\tau_{policy}$, τ_{action}, $\Delta\epsilon$, $maxSize$, σ_{task}, $\sigma_{solution}$)
    Init π_Ω: $Q_\Omega = 0, \forall s \in S, a \in A$ (Eq.(11))
    Init $\mathcal{L}_{train} \leftarrow RETRIEVE(\mathcal{L}, \langle \Omega, \pi_\Omega \rangle, maxSize, \sigma_{task})$
    Init ϵ (Eq. (12))
    Set $P_{policy}(.) \leftarrow$ POLICY_EVAL($\mathcal{L}_{train}$, τ_{policy})
    for each training episode $k = 1$ to K **do**
        for each step $h = 1$ to H **do**
            Observe state s
            Select $a \leftarrow \Pi$-Inference(s, $\mathcal{L}_{train}$, $P_{policy}(.)$, τ_{action}, ϵ)
            Perform a
            Observe reward R and follow-up state s'
            Update $Q_\Omega(s, a) \leftarrow Q_\Omega(s, a) + \alpha[R + \gamma \max_a Q_\Omega(s', a) - Q_\Omega(s, a)]$
            if goal state reached **then** end for
        if k is evaluation episode **and** $|\mathcal{L}_{train}| > 1$ **then**
            Update $\tau_{policy} \leftarrow \tau_{policy} + \Delta\tau_{policy}$
            Set $P_{policy}(.) \leftarrow$ POLICY_EVAL($\mathcal{L}_{train}$, τ_{policy})
        Update $\epsilon \leftarrow \epsilon - \Delta\epsilon$
    Update $\mathcal{L} \leftarrow RETAIN(\mathcal{L}, \langle \Omega, \pi_\Omega \rangle, \sigma_{solution})$

Algorithm 2 requires a policy library $\mathcal{L}$, that may be empty, a target task Ω, a temperature factor τ_{policy} for the policy importance, its increment value $\Delta\tau_{policy}$,

a temperature factor τ_{action} for the action probabilities, the decrement value $\Delta\epsilon$ for the exploration factor ϵ, the maximum training-library size $maxSize$, and the thresholds σ_{task} and $\sigma_{solution}$ to define similarities between tasks and solutions respectively.

It starts by initializing the Q-table for the policy π_Ω. It then selects cases for the training library $\mathcal{L}_{train}$ from all available cases as described in Algorithm 3. The *RETRIEVE* function takes as input the library $\mathcal{L}$, the target task Ω, the initialized solution π_Ω, and the maximum size $maxSize$ of $\mathcal{L}_{train}$, as well as the similarity threshold σ_{task}. First, it adds the target task/policy pair to a temporary library $\mathcal{L}_{temp}$ and then adds further cases to this library, according to the similarity function sim_{task} and the threshold σ_{task}, sorted by the similarity value. Finally, it returns the $maxSize$ most similar cases from $\mathcal{L}_{temp}$ and assigns them to $\mathcal{L}_{train}$. Notice that the target task is retained in $\mathcal{L}_{train}$ and that dissimilar tasks are simply ignored, avoiding *negative transfer*. However, for that we assume that the designer-specified similarity function sim_{task} provides a good estimation of task similarities.

Then we initiate ϵ, a probability which guides the training between exploration (a random policy) and exploitation (the current training policy), according to

$$\epsilon = \frac{1 + maxSize - |\mathcal{L}_{train}|}{maxSize}, \tag{12}$$

where $maxSize$ is the maximum size of $\mathcal{L}_{train}$. In this way, the starting value of ϵ dynamically takes the amount of knowledge into account that is associated with the size of the training library.

Algorithm 3 *RETRIEVE*

```
1: function RETRIEVE(L, ⟨Ω, π_Ω⟩, maxSize, σ_task)
2:     L_temp ← ⟨Ω, π_Ω⟩
3:     for ∀ϑ ∈ L do
4:         if sim_task(Ω, Ω_ϑ) ≥ σ_task then
5:             Insert ϑ in L_temp sorted by similarity
6:     if |L_temp| > maxSize then
7:         Remove the last |L_temp| − maxSize cases from L_temp
8:     return L_temp
```

The next step calls the policy evaluation function POLICY_EVAL. The main purpose of this function is to define a probability of using the policy from each case in $\mathcal{L}_{train}$. We propose to use the function in Algorithm 4, which takes the training library $\mathcal{L}_{train}$, the current target task Ω, and the policy temperature factor τ_{policy} as input. The function calculates an average reward $\bar{R}_\vartheta$ for every case ϑ in $\mathcal{L}_{train}$ during a testing phase and uses it to calculate the importance

factor of each case ϑ using a *softmax* function [4] defined as

$$P_{policy}(\vartheta) = \frac{e^{-\bar{R}_{\vartheta} * \tau_{policy}}}{\sum_{\vartheta_i \in \mathcal{L}_{train}} e^{-\bar{R}_{\vartheta_i} * \tau_{policy}}}. \tag{13}$$

It calculates the importance factors $P_{policy}(.)$ for all cases in the library $\mathcal{L}_{train}$. This way, we are tracking the usefulness of each policy in $\mathcal{L}_{train}$ for the current target task.

Algorithm 4 Policy evaluation

```
1: function POLICY_EVAL(L_train, Ω, τ_policy)
2:     for ∀ϑ ∈ L_train do
3:         R̄_ϑ ← Evaluate π_{Ω_ϑ} on Ω
4:     for ∀ϑ ∈ L_train do
5:         Calculate P_policy(ϑ) (Eq. (13))
6:     return P_policy(.)
```

The training starts in line 6 and performs K training episodes with a limited amount of steps H per episode. In each step the algorithm observes the current state s from the environment and then calls the Π-Inference algorithm (Algorithm 5) using this state s, the training library $\mathcal{L}_{train}$, the policy importance factors $P_{policy}(.)$, the policy temperature factor τ_{policy}, and the ϵ probability as input to select the action a the agent will take in the current state.

Algorithm 5 Combined *REUSE*/*REVISE*

```
1: function Π-INFERENCE(s, L_train, P_policy(.), τ_action, ϵ)
2:     With probability ϵ do
3:         return random action a_random
4:     for ∀ϑ ∈ L_train do
5:         Get action probabilities P_{action ϑ}(s, a_i) using τ_action (Eq. (14))
6:         Get weighted action probabilities P_{weighted ϑ}(s, a_i) using P_policy(.) (Eq. (15))
7:     Get final action probabilities P_final(s, a_i) (Eq. (16))
8:     for ∀a_i ∈ A do
9:         With probability P_final(s, a_i) do
10:            return a_i
```

Instead of following a single policy like in the ϵ-greedy strategy, Π-Inference blends the policies of the training library according to their importance for the current task. The function replaces separate *REUSE* and *REVISE* stages because it dynamically adjusts a gradual switch from reusing past knowledge to refining only the current policy. Here, we are returning a random action with a probability of ϵ and blending the policies in the training library otherwise. For

that we retrieve the Q-values $Q_{\Omega_\vartheta}(s, a_i)$ for each action a_i in the given state s for the policies of each case π_{Ω_ϑ} and transform them into action probabilities $P_{action\vartheta}(s, a_i)$ according to another *softmax* function defined as

$$P_{action\vartheta}(s, a_i) = \frac{e^{Q_{\Omega_\vartheta}(s,a_i)\frac{1}{\tau_{action}}}}{\sum_{a \in A} e^{Q_{\Omega_\vartheta}(s,a)\frac{1}{\tau_{action}}}}. \tag{14}$$

We then calculate the weighted action probabilities $P_{weighted\vartheta}(s, a_i)$ for each policy

$$P_{weighted\vartheta}(s, a_i) = P_{action\vartheta}(s, a_i) * P_{policy}(\vartheta) \tag{15}$$

and sum those probabilities in the next step to get the final action probabilities $P_{final}(s, a_i)$ for each action

$$P_{final}(s, a_i) = \sum_{\vartheta \in \mathcal{L}_{train}} P_{weighted\vartheta}(s, a_i). \tag{16}$$

We can then select one of the actions a_i according to the final probabilities. This makes sense because by using a stochastic approach for the behaviour selection of the learning agent, one can get rid of restrictions with regard to the used algorithm for training the policies of the cases in $\mathcal{L}$ and also different reward functions that could exist for different tasks.

The policy probability $P_{policy}(.)$ is re-evaluated in regular intervals, as described before, after performing an update of the policy temperature factor τ_{policy} with $\Delta\tau_{policy}$, to get the current importance distribution over the policies in the training library, progressively switching the attention to the new policy.

A training episodes ends after deciding if the learned target task Ω and its solution π_Ω will be added to the general library $\mathcal{L}$ as a new case ϑ in Algorithm 6. Algorithm 6 takes the general library $\mathcal{L}$, the training library $\mathcal{L}_{train}$, the target task Ω, its solution π_Ω, and the similarity threshold factor $\sigma_{solution}$ as input and returns the updated library $\mathcal{L}$. The function checks if one of the solutions π_{Ω_ϑ} in the training library is similar to the new solution π_Ω with respect to the threshold factor. Only if there is no similar solution the new case is added to the general library $\mathcal{L}$.

Algorithm 6 *RETAIN*

```
1: function RETAIN(L, L_train, ⟨Ω, π_Ω⟩, σ_solution)
2:     for ∀ϑ ∈ L_train do
3:         if sim_solution(π_Ω, π_Ωϑ) ≥ σ_solution then
4:             return L
5:     return L ∪ ⟨Ω, π_Ω⟩
```

4 Experimental evaluation

The domain and experiments we describe here aim at showing the viability of our approach when reusing knowledge in a simple domain.

4.1 Domain and experimental setup

The domain has been adapted from the original *Probabilistic Policy Reuse* publication [6], where an agent has to find its way to a goal position in an *office domain* consisting of rooms and connecting corridors represented in a grid world, as illustrated in Figure 2. The agent can move in four directions (left, right, up, down), but if it would walk against a wall it does not move and remains in the same position. When the agent reaches the goal state it receives a reward of $R = 1.0$, in all other states the reward is $R = 0.0$. There is only one goal state per task.

Although simple, this domain has been used in many publications (e.g. [10, 20, 28]) to offer a clear way to compare different *RL* algorithms without the need of much expert knowledge. Here, a task Ω is defined by its goal position and for each task only one goal is present and learned individually. The wall positions and size of our grid world are kept fixed during all experiments and we used the same settings for all approaches. We train every task for $K = 2000$ episodes (where an episode ends when the goal state is reached or a maximum of $H = 100$ steps is performed) and evaluate the policy quality at each step by trying to solve the task from 10 different positions, as shown in Figure 2 (right) averaging over 10 episodes. We use the same setting for estimating the importance factor $P_{policy}(.)$ of each policy in $\mathcal{L}_{train}$ when learning with *CBPI* while updating the policy temperature factor τ_{policy} every 40 episodes. The task similarity is determined as the euclidean distance between goal states and the selection is limited by a percentage of the grid diagonal. Although primitive, this method already provides good results in this domain.

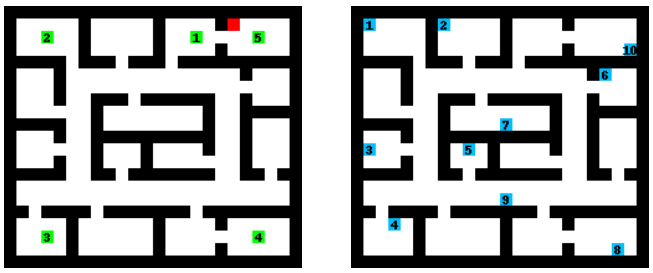

Fig. 2. The used grid world domain for a simple robot navigation scenario in the experiment. The graphs show all goal positions that were considered as individual tasks (green: previously solved tasks, red: task to learn) (left) and the starting positions of the agent during evaluation (blue) (right).

We perform two experiments. The first one shows the suitability of our approach for learning a new task with an existing library, and the second one shows that it can also be used for building a library with core policies for a domain from scratch. The code and logs of all results reported here are documented and available online[1].

4.2 Learning a new policy with CBPI

In this experiment we evaluate our *CBPI* algorithm against *Policy Library Policy Reuse (PLPR)* [6] and vanilla *Q-Learning*.

For this purpose, we let an agent train on five source tasks $\Omega_{1,\dots,5}$ using *Q-Learning* until an optimal policy is learned, and save the learned policies $\pi_{\Omega_1,\dots,\Omega_5}$ for each case in the library $\mathcal{L}$. The agent then has to learn to perform well on a new target task Ω and learn a policy π_Ω for that task. The goal positions of the source tasks (green) and the target task (red) are shown in Figure 2 (left).

As a baseline, we trained the target task Ω from scratch using *Q-Learning* with a learning rate $\alpha = 0.05$ and a discount factor $\gamma = 0.95$. The strategy used to balance between exploration and exploitation was the $\epsilon - greedy$ strategy with a starting value of $\epsilon = 1.0$ for maximal exploration and a decay value of $\Delta\epsilon = 0.0005$, which was subtracted after each episode until the end of training.

While *PLPR* always starts the training of a new task with all available policies in the library, the *RETRIEVE* stage of *CBPI* selects only the most promising policies through a similarity measure sim_{task}. For this domain, sim_{task} is defined by the euclidean distance between the goal positions of each task which, despite simple, works very well. Here, we only select the $maxSize = 3$ most similar tasks to guide the training including the task we want to learn. During training we follow an ϵ-greedy strategy, but since we assume that the chosen policies are similar to the one we want to build, we initialize ϵ as described in 12 to better exploit past knowledge, but instead of choosing between *random* and *greedy*, we choose between *random* and the Π-Inference strategy as described in Section 3.

We evaluate two scenarios with the same training settings. In the first scenario, we are using all available policies for the policy library, $\mathcal{L}_{1,2,3,4,5} = \{\pi_{\Omega_1}, \pi_{\Omega_2}, \pi_{\Omega_3}, \pi_{\Omega_4}, \pi_{\Omega_5}\}$. In the second scenario, we deliberately chose to only use policies from tasks that seem very unrelated to our target task, so the policy library becomes $\mathcal{L}_{2,3,4} = \{\pi_2, \pi_3, \pi_4\}$. The results provided here are averages of 50 runs for each task.

In Figure 3 we can see the development of the policy importance factors $P_{policy}(.)$ for all policies in $\mathcal{L}_{train}$ over training episodes for *CBPI* (left), and the probabilities for the *PLPR* approach (right). For *CBPI*, the policies have a very similar importance at the beginning, but the current policy becomes rapidly dominant and has the greatest influence on the training performance. As soon as a policy reaches the importance threshold of 0.2 it gets *deactivated* and won't be considered anymore until the end of training. The *PLPR* algorithm does not evaluate the policies specifically, but calculates a W-value after every episode

[1] openly available under MIT License: https://github.com/cowhi/CBPI

that indicates the usefulness of a policy for the current task according to the results achieved. At the beginning of every episode, the W-values are transformed into a probability to select the according policy and then a policy is chosen as guiding policy for the next episode based on this probability, Figure 3(right).

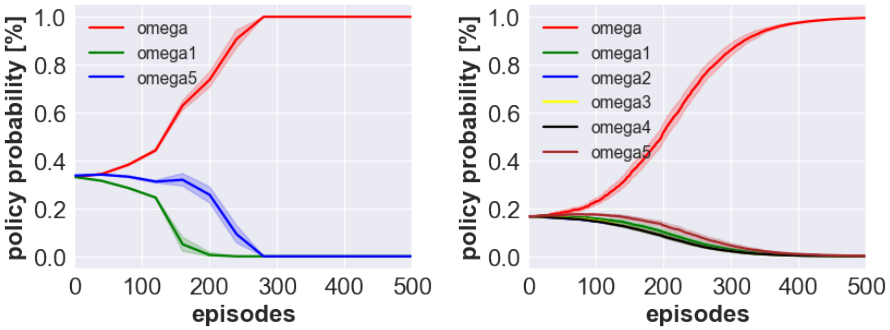

Fig. 3. Development of policy probabilities $P_{policy}(.)$ for policies in the training library $\mathcal{L}_{Train} = \{\pi_{\Omega_1}, \pi_{\Omega_5}, \pi_{\Omega}\}$ when using CBPI (left) and when using PLPR with all policies $\pi_{\Omega_{1-5}}$(right).

The first part of this experiment uses a library $\mathcal{L}$ that contains 5 cases corresponding to tasks $\Omega_{1,\ldots,5}$ including two, $\Omega_{1,5}$, that are very similar to the target task Ω. In Figure 4 we can see the achieved rewards per episode and both *PLPR* and *CBPI* outperform the standard *Q-Learning* approach by far. It is also notable, that our algorithm performs better at the beginning of the training, due to the fact that it evaluates the available policies before it starts training, while *PLPR* starts randomly and learns the weights W for each policy on the fly.

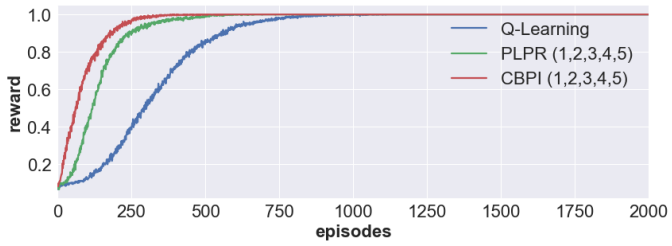

Fig. 4. Resulting rewards when using a policy library that also contains policies from very similar tasks (Ω_1 and Ω_5).

The second scenario uses a library $\mathcal{L}$ that contains 3 cases corresponding to tasks $\Omega_{2,3,4}$ without the policies that are very similar to the target task Ω.

Again, the *PLPR* approach uses all available tasks for its training library $\mathcal{L}_{train}$, while the similarity metric in *CBPI* detects that there are no similar tasks in the library and builds the training library ignoring the available cases. In Figure 5 we can see the achieved rewards per episode. Here, it is interesting to see that *CBPI* and *Q-Learning* perform exactly the same, meaning that the unhelpful policies were successfully detected, while *PLPR* takes much longer to converge suffering from *negative transfer*.

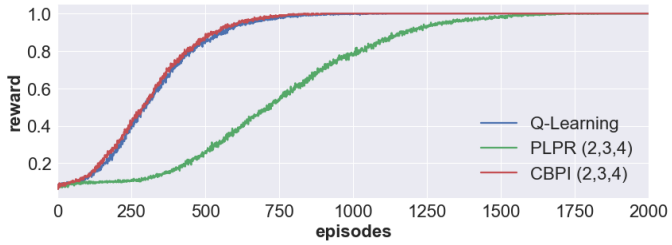

Fig. 5. Resulting rewards when using a library only containing policies from very unrelated tasks (Ω_2,Ω_3, and Ω_4).

The results shown above show that our approach performs slightly better than *PLPR* when the available knowledge contains similar task, and is clearly better if the available knowledge only contains unrelated tasks. *CBPI* has shown that it benefits from existing favorable knowledge while in the worst case performs as if no *a priori* knowledge was available. This behaviour shows that it can better deal with the problem of *negative transfer* during training, as long as a well-defined similarity metric is given.

4.3 Building a core policy library with *CBPI*

The grid in Figure 6 (left) shows the locations of 50 goal positions that we used in our experiment to build a core policy. The results are shown for *CBPI* (middle) and *PLPR* (right). It is clearly visible that both algorithms have similar behaviour and select a good set of policies. It is however notable, that *CBPI* provides a set of solutions that contains a policy for every room apart from the connecting rooms, which makes sense since those can already be reached by the policy for the next room. The *PLPR* on the other hand seems to put an emphasize on those connecting rooms and does not offer solutions for two of the rooms in the middle of the grid. We therefore conclude that our selection approach in the *RETAIN* stage is sensitive enough for selecting core policies in this domain and provides acceptable results compared to *PLPR*.

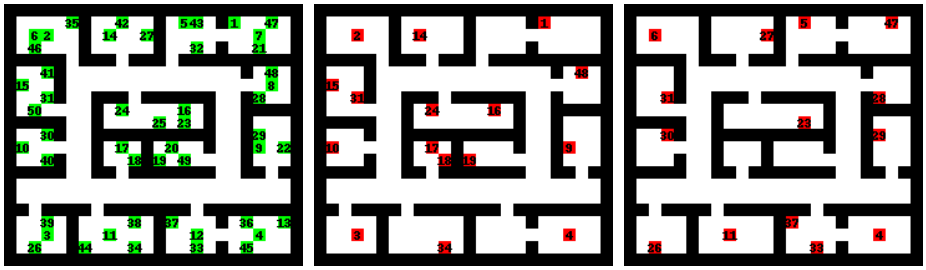

Fig. 6. The same grid world domain was evaluated for the library building experiment. The graphs show all 50 goal positions that were learned during the experiment (left) and the extracted core policies using *CBPI* (middle) and *PLPR* (right) respectively.

5 Related work

The most closely related work is *PLPR* as introduced by Fernandéz & Veloso [6]. The authors propose a framework to autonomously build a library of core policies for a given domain. The approach differs from ours in the way that we are blending policies during training according to usefulness for the current task at every step, but only as long as it seems beneficial, where the other approach selects a whole policy to be followed for a certain amount of time.

Sharma et al. [19] follow another approach to combine *CBR* and *RL*, proposing a *CBR* framework that is used as an instance-based state function approximator in a layered *RL* architecture. Similarly, Gabel & Riedmiller [7] also make use of *CBR* for function approximation. We, on the other hand, directly use previously learned policies for accelerating learning in the new task.

Bianchi et al. [3] also rely on a *CBR* approach to accelerate learning. However, their focus is on extracting heuristics from the source tasks, rather than explicitly reusing policies as we do.

Koga et al. [10] propose to blend multiple policies into a single abstract policy, which is used at the beginning of learning in any new task (whether the new task is similar to the source tasks or not). In spite of following a similar idea, *CBPI* stores multiple concrete policies, and selects only the most promising ones by taking similarity with the target task into account.

Sinapov et al. [22] evaluate user-defined task features so as to enable the estimation of the similarity between tasks and choose only the most similar one(s) for the target task. However, they are more focused on source-task selection, and the transfer is simply accomplished by reusing value functions from one task to another. *CBPI* on the other hand is focused on providing a consistent framework to select and reuse the source policies as good as possible.

6 Conclusions and future work

We here proposed a framework to integrate *Case-Base Reasoning* with *Reinforcement Learning*. Building on this framework we introduced *Case-Based Policy*

Inference which consistently reuses gathered knowledge from previously learned tasks in order to accelerate learning of a new target task. It exploits previously learned policies from cases that are similar to the target task and blends the policies during training, progressively switching the control to the target policy.

We compared *CBPI* with *PLPR* and *Q-Learning*, and showed that our proposal learns the target task faster and is more robust to negative transfer. We have also shown that *CBPI* is effective when building a library of core policies for a domain.

The results reported here indicate that applying the *CBR* methodology on *RL* is a promising approach. Further works will focus on implementing the framework with more sophisticated *RL* algorithms and evaluate it in more complex domains.

Acknowledgments

We are grateful for the support from CAPES, CNPq (grant 311608/2014-0), and FAPESP (grants 2015/16310-4, 2016/ 21047-3). We also thank Google (Research Award) and the Nvidia corporation (GPU donation).

References

1. Aamodt, A., Plaza, E.: Case-based reasoning: Foundational issues, methodological variations, and system approaches. AI communications 7(1), 39–59 (1994)
2. Aha, D.W.: The omnipresence of case-based reasoning in science and application. Knowledge-based systems 11(5), 261–273 (1998)
3. Bianchi, R.A., Ros, R., De Mantaras, R.L.: Improving reinforcement learning by using case based heuristics. In: Proceedings of the 8th International Conference on Case-Based Reasoning (ICCBR). pp. 75–89. Springer (2009)
4. Bridle, J.S.: Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In: Proceedings of the 2nd International Conference on Neural Information Processing Systems. pp. 211–217. MIT Press (1989)
5. Cheetham, W., Watson, I.: Fielded applications of case-based reasoning. The Knowledge Engineering Review 20(03), 321–323 (2005)
6. Fernández, F., Veloso, M.: Probabilistic policy reuse in a reinforcement learning agent. In: Proceedings of the 5th International Conference on Autonomous Agents and Multiagent Systems (AAMAS). pp. 720–727 (2006)
7. Gabel, T., Riedmiller, M.: Cbr for state value function approximation in reinforcement learning. In: Proceedings of the 6th International Conference on Case-Based Reasoning (ICCBR). pp. 206–221. Springer (2005)
8. Glatt, R., da Silva, F.L., Costa, A.H.R.: Towards knowledge transfer in deep reinforcement learning. In: Proceedings of the 5th Brazilian Conference on Intelligent Systems (BRACIS). pp. 91–96. IEEE (2016)
9. Hullermeier, E.: Credible case-based inference using similarity profiles. IEEE Transactions on Knowledge and Data Engineering 19(6), 847–858 (2007)
10. Koga, M.L., Freire, V., Costa, A.H.: Stochastic abstract policies: Generalizing knowledge to improve reinforcement learning. Cybernetics, IEEE Transactions on 45(1), 77–88 (2015)

11. Kolodner, J.: Case-based reasoning. Morgan Kaufmann (2014)
12. Konidaris, G., Scheidwasser, I., Barto, A.G.: Transfer in reinforcement learning via shared features. Journal of Machine Learning Research (JMLR) 13(1), 1333–1371 (2012)
13. Kuhlmann, G., Stone, P.: Graph-based domain mapping for transfer learning in general games. In: Proceedings of the 18th European Conference in Machine Learning (ECML). pp. 188–200. Springer (2007)
14. McCallum, R.A.: Instance-based utile distinctions for reinforcement learning with hidden state. In: Proceedings of the 12th International Conference on Machine Learning (ICML). pp. 387–395 (1995)
15. Mnih, V., Silver, D., Rusu, A.A., Riedmiller, M., et al.: Human-level control through deep reinforcement learning. Nature 518(7540), 529–533 (2015)
16. Ng, A.Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., Liang, E.: Autonomous inverted helicopter flight via reinforcement learning. In: Experimental Robotics IX, pp. 363–372. Springer (2006)
17. Pan, S.J., Yang, Q.: A survey on transfer learning. Knowledge and Data Engineering, IEEE Transactions on 22(10), 1345–1359 (2010)
18. Puterman, M.L.: Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons, Hoboken, NJ, USA (2014)
19. Sharma, M., Holmes, M.P., Santamaría, J.C., Irani, A., Isbell Jr, C.L., Ram, A.: Transfer learning in real-time strategy games using hybrid cbr/rl. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI). vol. 7, pp. 1041–1046 (2007)
20. Sherstov, A.A., Stone, P.: Function approximation via tile coding: Automating parameter choice. In: Abstraction, Reformulation and Approximation, pp. 194–205. Springer (2005)
21. Silva, F.L.d., Glatt, R., Costa, A.H.R.: Simultaneously learning and advising in multiagent reinforcement learning. In: Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS) (2017)
22. Sinapov, J., Narvekar, S., Leonetti, M., Stone, P.: Learning inter-task transferability in the absence of target task samples. In: Proc. 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS). pp. 725–733 (2015)
23. Stone, P., Sutton, R.S.: Scaling reinforcement learning toward robocup soccer. In: Proceedings of the 18th International Conference of Machine Learning (ICML). pp. 537–544. ACM (2001)
24. Sutton, R.S., Barto, A.G.: Introduction to Reinforcement Learning. MIT Press, Cambridge, MA, USA (1998)
25. Taylor, M.E., Stone, P.: Transfer learning for reinforcement learning domains: A survey. Journal of Machine Learning Research (JMLR) 10, 1633–1685 (2009)
26. Tesauro, G.: Temporal difference learning and td-gammon. Communications of the ACM 38(3), 58–68 (1995)
27. Thrun, S., Mitchell, T.M.: Lifelong robot learning, vol. 15. Elsevier (1995)
28. Thrun, S., Schwartz, A.: Finding structure in reinforcement learning. Proceedings of the 7th International Conference on Neural Information Processing Systems (NIPS-94) pp. 385–392 (1995)
29. Watkins, C.J., Dayan, P.: Q-learning. Machine Learning 8(3-4), 279–292 (1992)
30. Watson, I.: Case-based reasoning is a methodology not a technology. Knowledge-based systems 12(5), 303–308 (1999)